

ImageMagick Examples -- Animation Basics

Index

▶ [ImageMagick Examples Preface and Index](#)

▶ [GIF Animations and Animation Meta-data](#)

▶ [Frame Disposal Methods](#)

- [Dispose None](#) - overlay each frame in sequence
- [Dispose Previous](#) - preserve background canvas
- [Dispose Background](#) - clear to background

▶ [Studying Animations](#)

- [Identify](#) - information about an animation
- [Adjoin](#) - splitting into individual frame images
- [Coalesce](#) - fill out frames completely
- [Frame Montage](#) - the "gif_anim_montage" script
- [List Information](#) - rebuild an existing animation
- [Disposal Images](#) - the GIF dispose form of the frames
- [Deconstruct](#) - report areas of frame differences
- [Frame Comparisons](#) - more detailed frame differences
[Compare Any](#), [Compare Clear](#), [Compare Overlay](#)

▶ [Types of Animations](#)

- [Coalesced Animations](#)
- [Overlay Animations](#)
- [Cleared Frame Animations](#)
- [Mixed Disposal Animations](#)

▶ [The End of the Loop](#) - when an animation stops running

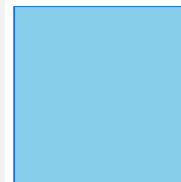
▶ [Zero Delay Intermediate Frames](#)

These examples continue the previous example page on [Layers of Multiple Images](#) but instead of layering multiple images on top of each other to produce a single image, here we display each image for a short period of time so as to produce an animation of images. The following section provides a basic understanding of the complexities of animations and specifically GIF animations. It looks at the basic methods used to generate animations, and how you can study existing animations to get an understanding of how they work. This is recommended reading before going further in any of the later animation sections.

GIF Animations and Animation Meta-data

The default way ImageMagick handles the output of an image list is to generate a multi-page image. For the GIF image format, however, this takes the special form of a 'GIF animation'.

```
magick -delay 100 -size 100x100 xc:SkyBlue \
    -page +5+10 balloon.gif -page +35+30 medical.gif \
    -page +62+50 present.gif -page +10+55 shading.gif \
    -loop 0 animation.gif
```



Here is a more advanced 'sparkle' example that uses a shell script "[star_field](#)". This script was developed from my experiments in generating [random star fields](#).

```
star_field 70x46 stars1.gif
star_field 70x46 stars2.gif
star_field 70x46 stars3.gif
magick rose: -compose Screen \
    \( -clone 0 stars1.gif -composite \) \
    \( -clone 0 stars2.gif -composite \) \
    \( -clone 0 stars3.gif -composite \) \
    -delete 0 -set delay 25 -layers Optimize rose_sparkle.gif
rm stars[123].gif
```



Basically three random star fields are generated, at the right size, then overlaid onto our image, the IM built-in "[rose:](#)", using a '[Screen](#)' alpha composition to brighten the image with the given star patterns. The whole thing is then run through the IM general GIF animation optimizer. The above may seem complex as it is using some advanced IM features I have yet to introduce, but the result is a relatively simple, but well optimized three frame animation. You can also look at some of the more complex animations that were created using simple shell scripts for [Distortion Animations](#). There are a few extra IM settings which were created specifically for use in GIF animations, and knowing about these is the first step into the world of GIF animations...

[-dispose](#) {method}

What the following images should do with the previous results of the GIF animation. Valid options are '[Undefined](#)', '[None](#)', '[Previous](#)', and '[Background](#)'. (See below for explanation of the settings)

[-loop](#) {number} Number of times the GIF animation is to cycle through the image sequence before stopping. It is an output 'image write' setting, so can be set anywhere on the command line, though only the last such setting will be used. Usually this set by default, to zero (infinite loop), however if any image read in has a different value, then this setting will be set to that images value. As such I recommend that you always set "[-loop](#)" when creating a GIF animation, after all the images has been read in. For more information see [The End of the Loop](#) below.

[-delay](#) {time} Set the time delay (in 1/100th of a second) to pause after drawing the images that are read in or created after this setting has been defined. You can specify a different scale for the time delay by specifying a '[x](#)' scaling (giving in ticks per second). For example '[10x1](#)' is 10, 1 second ticks, while '[10x100](#)' is 10, one hundredth of a second ticks. Basically the '[x](#)' is equivalent to a fraction '[/](#)' sign. For example if you specify '[1x160](#)' will set a delay that is appropriate for 160 frames per second.

 GIF animation delays must be specified in hundredths of a second for correct working, which is why that is the default time unit. The 'x' factor is used more for generating other more movie like formats, such a MNG's, and AVI's.

-set dispose {method}

-set delay {time}

While the previous option settings will set image attributes on newly created, or image that are read in, *after* that option is given, the "**-set**" option is an operator, that will allow you set image attributes on all images that have *already* in the current image sequence. This allows you to change the setting over a whole animation, or just a single frame, after the images have been loaded or modified.

-page {w}x{h}+{x}+{y}

This lets you set the offset position of the image about to be read in. As this is a setting option, it only applies the geometry you give to images that follow the setting. It does not effect images already read into memory.

If not given, or turned off using "**+page**" the offset for the image read will be preserved. If the image does not have an offset it will be positioned at '+0+0' or the top left corner of the working canvas or 'page'.

It can also be used to define a larger working canvas, by specifying a width 'x' height. Only the width and height page setting of the first image in the sequence will be used to set the overall GIF animation canvas size, all other page size settings will be ignored when the animation is finally written. When a GIF animation is read in the canvas size is set on all the frames in the animation. MNG animations can save frame offsets, but does not save canvas sizes. The size of the first image defines the canvas size of the whole animation.

 The GIF image format can not specify a negative offset for images on a canvas. If you try to use a negative offset IM will reset it to zero when that image (or animation frame) is written to a GIF file.

Positive offsets larger than the image canvas are quite acceptable but may result in the image not appearing in the canvas drawing area when magick displayed. How a GIF animation display program handles this is undefined. Caution is advised.

-repage {w}x{h}+{x}+{y}

This is exactly like "**-page**" except that it is an image operator instead of a setting. That means you can use this to change or reset the 'page geometry' of an image or animation frame that has already being read into memory. The simpler "**+repage**" form, just resets the 'page geometry' of all images to the actual image in each frame in the current image sequence to a zero offset, and the images actual size. This operation is vital when you are extracting the individual frames from an animation, (See the [Adjoin Examples](#) below). However "**+repage**" will destroy a lot of positioning information stored in each image, as such you should also probably extract this information into a separate file for later re-use. See [Animation List Information](#) below.

Important Point

DO NOT save the intermediate, animations which you are not finished processing, directly to GIF. You can use the IM internal format MIFF, as a temporary file format, if you want to work on an animation in series of separate processing steps. I repeat...

Do not use GIF as an intermediate file format, use MIFF instead

If you made the big mistake of saving to GIF you would have just made the resulting animation worse, as IM would have now performed an automatic [Color Quantization](#), to reduce the number of colors present. Not only that but it did so on each frame completely independently to every other frame, making any further processing, particularly any GIF optimizations just that much harder. Solving this is a complex, multi-level problem, which is looked at in the next section [Animation Optimization](#).

Frame Disposal Methods

The first thing people creating GIF animation have trouble with is the "[-dispose](#)" setting. This is not surprising as it is a complex setting. Worse still a lot of animation programs, including many web browsers, don't always handle the GIF disposal meta-data setting correctly. However using the right disposal can make a big difference to how well your animation works and optimizes. The first thing to remember in ImageMagick is that almost all the special animation options, are settings for image reading. That is, they are applied to images that are read in, after, the setting has been given. The "[-loop](#)" setting is the only one typically used after the animation has been completed, just before the the animation is saved. The basic task of the "[-dispose](#)" defines how an image is to be removed, *after* it has been displayed for its "[-delay](#)" time period. That is, you need to give an image's "[-dispose](#)" and "[-delay](#)" settings *before* reading the image for that frame. But the action is applied after that image is displayed. This is a little counter intuitive but does make sense in the way IM operates on images. If you remember this, you should have no problems. The 'plus' forms of these options, like most other settings in IM stops the setting being applied to any images being read in. That means if you don't specify a setting, the frame image will continue to use the setting that was read in with the image (if any). This can be important later when you want to read in a GIF animation for further processing. Or when merging one GIF animation into another (the most difficult animation technique).

Dispose None - each frame overlaid in sequence

The default "[-dispose](#)" setting for GIF animations is 'Undefined' which most animation programs treats the same as a 'None' disposal setting. Basically this tells the computer to just leave whatever is overlaid by this specific frame. Or more precisely, 'do nothing'. However please note that the whole canvas is always cleared at the end of the animation sequence, before it loops and repeats. Here for example is a standard 'None dispose' animation...

```
magick -delay 100 -dispose None \  
    -page 100x100+5+10 balloon.gif \  
    -page +35+30 medical.gif \  
    -page +62+50 present.gif \  
    -page +10+55 shading.gif \  
    -loop 0 anim_none.gif
```



This disposal technique is ideal for animations which involve no form of transparency, such as animations drawn on a solid, or patterned background.

```
magick -dispose none -delay 100 \
      -size 100x100 xc:SkyBlue +antialias \
      -fill DodgerBlue -draw 'circle 50,50 15,25' \
      -page +5+10 balloon.gif \
      -page +35+30 medical.gif \
      -page +62+50 present.gif \
      -page +10+55 shading.gif \
      -loop 0 canvas_none.gif
```



Note that this technique can only add visible colors to an animation. It can never actually make any part of an animation transparent again. (See [Overlay Animations](#) below). To also handle transparency need to use one of the other sorts of disposal methods.

Dispose Previous - preserve background canvas

The '**Previous**' disposal method is relatively simple. When the current image is finished, return the canvas to what it looked like before the image was overlaid. If the previous frame image also used a '**Previous**' disposal method, then the result will be that same as what it was before that frame.. etc.. etc.. etc... For example in this animation each of the later frames will return to the very first frame of the image, which has a '[None](#)' disposal setting, before overlaying the image associated with that frame. The result is a background canvas that has just each frame image overlaid for just the duration of that image...

```
magick -dispose none -delay 0 \
      -size 100x100 xc:SkyBlue +antialias \
      -fill DodgerBlue -draw 'circle 50,50 15,25' \
      -dispose previous -delay 100 \
      -page +5+10 balloon.gif \
      -page +35+30 medical.gif \
      -page +62+50 present.gif \
      -page +10+55 shading.gif \
      -loop 0 canvas_prev.gif
```

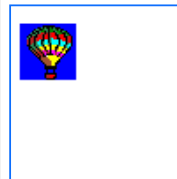


Note the "[-dispose](#)" method '[None](#)' used for the first image. This is important, otherwise the 'previous' frame will go all the way back to the original empty canvas that was present before the first frame. Also note that I used a "[-delay](#)" of '0' in the above animation. This says not to wait before overlaying the first frame onto this 'background canvas'. Without it you will see a short delay, showing just the canvas image with nothing on top of it. Of course I need to still set a longer "[-delay](#)" for the later images, or they will appear and disappear in the wink of an eye, and incidentally use up a lot of the viewers CPU cycles. The use of the '**Previous**' disposal method can be prone to a slight flickering, or pause in some web browsers, especially on slower machines. Though that is quite rarely seen these days, the flicker itself is still present, and something I consider to be a bug. See [Zero Delay Frames](#) below for more specifics. Few animations make use of a dispose previous style of animation, the reason is that it is very difficult for computers to optimise. The problem is just what frame should the computer pick to become the background image? Simple for us humans to figure out the best image to use, but difficult for a computer decide. The best background image to use in an animation may not even be meant to be displayed, such as in the current example, and as such may not exist in an un-optimized version of that animation.

Dispose Background - clear to background

While the first two "[-dispose](#)" methods are relatively simple, the '**Background**' is probably the hardest to understand. When the time delay is finished for a particular frame, the area that was overlaid by that frame is cleared. Not the whole canvas, just the area that was overlaid. Once that is done then the resulting canvas is what is passed to the next frame of the animation, to be overlaid by that frames image. Here for example we just replace each frame with the next frame.

```
magick -delay 100 -dispose Background \
      -page 100x100+5+10 balloon.gif \
      -page +35+30 medical.gif \
      -page +62+50 present.gif \
      -page +10+55 shading.gif \
      -loop 0 anim_bgnd.gif
```



So you can see exactly what is going on, lets add an initial canvas image to the animation, so you can see how a '**Background**' actually 'disposes' that frame from the animation display.

```
magick -delay 100 -dispose none \
      -size 100x100 xc:SkyBlue +antialias \
      -fill DodgerBlue -draw 'circle 50,50 15,25' \
      -dispose background \
      -page +5+10 balloon.gif \
      -page +35+30 medical.gif \
      -page +62+50 present.gif \
      -page +10+55 shading.gif \
      -loop 0 canvas_bgnd.gif
```



As you can see as each overlaid frame is disposed of, that frames area is cleared to transparency, before the next image is overlaid. This is the importance of this GIF disposal method as it is the only way GIF animations can clear any pixel regardless of an animations frame history. The only other way to clear pixels is to use '[Previous](#)' to go back to a frame in which those pixels were clear. But that relies on knowing the history of the animation sequence which makes it much more difficult for computers to optimize.



*There is some thinking that rather than clearing the overlaid area to the transparent color, this disposal should clear it to the 'background' color meta-data setting stored in the GIF animation. In fact the old "**Netscape**" browser (version 2 and 3), did exactly that. But then it also failed to implement the '**Previous**' dispose method correctly.*

On the other hand the initial canvas should also be set from the formats 'background' color too, and that is also not done. However all modern web browsers clear just the area that was last overlaid to transparency, as such this is now accepted practice, and what IM now follows.



*Before IM version 6.2.6-1, the IM "[-coalesce](#)" and "[-deconstruct](#)" operations did not handle animations that used '**Background**' disposal to make pixels transparent, as per all the major web browsers. See [Animation Bugs](#) for examples and details.*

These functions however did work fine when no pixel clearing was applied or intended. This has now been fixed for "[-](#)

[coalesce](#)" and the "[-layers OptimizeFrame](#)" method was created to replace the use of "[-deconstruct](#)" as a GIF animation frame optimizing function.

Studying Animations

Before we can continue with the basics of GIF animation, their types, optimizations, and handling techniques, we need some techniques for studying existing animations.

Identify - information about and animation

Now an animation consists of a lot of information packed into each individual frame. You can see some of this information using the default IM "[identify](#)" command.

```
magick identify canvas_prev.gif
```

```
canvas_prev.gif[0] GIF 100x100 100x100+0+0 8-bit PseudoClass 2c 1.42kb
canvas_prev.gif[1] GIF 32x32 100x100+5+10 8-bit PseudoClass 16c 1.42kb
canvas_prev.gif[2] GIF 32x32 100x100+35+30 8-bit PseudoClass 8c 1.42kb
canvas_prev.gif[3] GIF 32x32 100x100+62+50 8-bit PseudoClass 8c 1.42kb
canvas_prev.gif[4] GIF 32x32 100x100+10+55 8-bit PseudoClass 4c 1.42kb
```



If you did not see output like the above your IM is a little old, and you really should upgrade your installed version of ImageMagick, to the latest version. If you don't you will be missing out on a lot of the new advances in IM's handling and control of GIF animations.

As you can see the actual image saved for the second and later frames is only 32x32 pixels, but all the frames sits on a 100x100 pixel 'virtual canvas' with a 'virtual offset' on that larger canvas. To see more of the various bits of meta-data that is present you need to use some of the more specialized [percent Escape Formats](#) to get IM to output it.

```
magick identify -format "%f canvas=%Wx%H size=%wx%h offset=%X%Y %D %Tcs\n" \
    canvas_prev.gif
```

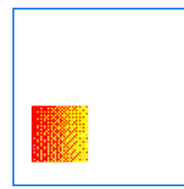
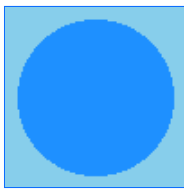
```
canvas_prev.gif canvas=100x100 size=100x100 offset=+0+0 None 0cs
canvas_prev.gif canvas=100x100 size=32x32 offset=+5+10 Previous 100cs
canvas_prev.gif canvas=100x100 size=32x32 offset=+35+30 Previous 100cs
canvas_prev.gif canvas=100x100 size=32x32 offset=+62+50 Previous 100cs
canvas_prev.gif canvas=100x100 size=32x32 offset=+10+55 Previous 100cs
```

Which clearly shows not only the canvas size, image size and offset, but also the disposal and time delays used for each individual frame. Note how the first frame has the different disposal and time delay that was needed for proper use of the later '[Previous](#)' disposal method.

Adjoin - splitting an animation into frames

Now as you saw above, ImageMagick will by default try to save multiple images into one file if that file format allows it. However as discussed in [Writing a Multi-Image List](#) IM will let you use the "[+adjoin](#)" setting to tell it to save each image to disk as a separate individual image. For example, here we read in one of the GIF animations and output the individual frame images in the animation sequence.

```
magick canvas_prev.gif -scene 1 +adjoin frame_%03d.gif
```

If you were to examine the actual images above you will find that although most web browsers show a larger 100x100 area, on which each sub-frame appears. In fact most of the actual images show are really only just 32x32 pixels, just as show in the previous 'identify' commands above. That is, most of the area is just a canvas on which nothing is drawn, known as the images 'page geometry' or 'virtual canvas'. The first image of the animation defines that larger 'canvas' and every other frame defines an 'offset' position on this larger canvas. This extra information is preserved in the frames that was saved by the ["+adjoin"](#) setting. As as such you can easily re-build the GIF animation. Not only is the page information preserved in each separate frame image, but also any delay, looping and GIF dispose settings, is also preserved. This means that to rebuild the animation you only need to read all the images in.

```
magick frame_???.gif anim_rebuilt.gif
```



Sometimes however you don't want to preserve this page geometry information. For example if you want to use the individual frames for other projects. You can reset the page size and offset using the ["+repage"](#) option, to remove the 'virtual canvas' information, leaving just the actual image.

Normally when extracting animation sub-images you also generally reset the images delay and dispose settings too to ensure they don't interfere with the editing and display. For example here I remove the unwanted virtual canvas and offset and reset the timing delays and disposals.

```
magick canvas_prev.gif +repage -set delay 0 -set dispose None \
+adjoin repage_%03d.gif
```



Of course if you junk that meta-data, you need some way of recording and editing that data. See [Animation List Information](#) (below) for a script that extracts both the sub-images and saves the animation meta-data, in a form that can be used to re-build the animation.

Coalesce - fill out frames completely

Viewing an animation in the form of the sub-frames, however is usually not very useful, in a typical animation. For one thing, a highly optimized animation can consist of lots of very small parts, without any visual indication of how they fit together. It can also have a lot of other 'noise' that was added for [Compression Optimization](#) to reduce the overall file size of the animation. For example, it is very difficult to figure out what this animation actually did, just by looking at the individual sub-frames of the animation.

```
magick script_k.gif +repage +adjoin script_k_%02d.gif
```




The ["-coalesce"](#) operation basically converts an image into exactly what the animation should look like after the previous frame has been correctly [disposed](#), and the next sub-frame overlaid. That is, instead of an animation sequence where each frame only represents the overlaid changes to the previous 'disposed' frame. This operator creates a complete view of the animation at each point, a bit like a true film strip, rather than an animation sequence. Such a sequence, known as a [Coalesced Animation](#) is much easier to study, edit, modify and re-optimize. Here for example will generate a montage of the same 'confusing' animation sequence I showed above, but this time we'll ["-coalesce"](#) the sequences, so you can see what is really happening.

```
montage script_k.gif -coalesce \  
  -tile x1 -frame 4 -geometry '+2+2' \  
  -background none -bordercolor none coalesce_k_montage.gif
```



As you can see the result is like a film strip of the animation, allowing you to clearly see how the previous pieces fit together to form a hand drawn letter 'K'. As of IM version 6.2.6, the `"magick montage"` command understood the use of ["-coalesce"](#), allowing you to create 'film strip' like image of the animation frames, exactly as shown above. This version also contained fixes for coalesce, and any GIF animation work should be at least this version (or better still the latest version).

 An even better montage technique for examining animations is given in the next example section.

The ["-dispose"](#) setting of a *coalesced image sequence* is actually irrelevant, in a [Coalesced Animation](#). However for users piece of mind the ["-coalesce"](#) operator will set the ["-dispose"](#) setting of each frame to either ['None'](#) or ['Background'](#) as appropriate, so that the coalesced image sequence will continue to animate correctly (as shown above).

 A frame with a ['Background'](#) disposal means the next frame needed to clear at least one or more pixels, to be displayed correctly.

As such animations in which ["-coalesce"](#) added a ['Background'](#) dispose, means that the animation can not be saved as a simple [Overlay Animation](#) (see below).

Technically, you can set all dispose settings of a coalesced image sequence to either ['Background'](#) or ['Previous'](#) to generate a [Cleared Frame Animation](#) (see below). Though not all animations will optimize well in that form.

There are also some non-animation uses of the ["-coalesce"](#) operator. See [Coalesce, and Progressive Flattening](#) examples of these uses.

Animation Frame Montage - the "gif_anim_montage" script

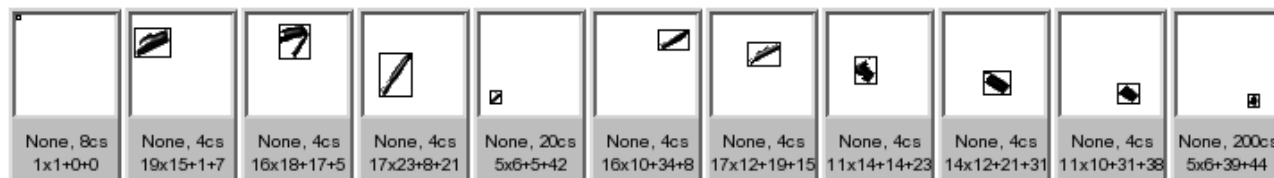
While ["+adjoin"](#) operator will let you extract the actual images from an animation and ["-coalesce"](#) will let you see the resulting frames of the animation, both methods leave out a lot of information about the animation. By using some very careful manipulation of the animation images, you can display the frames so as to show not only the actual frames, but also the placement of those frames on the larger canvas. Here is one such method of displaying an animation.

```
magick -dispose Background script_k.gif -alpha set \
  -compose Copy -bordercolor black -border 1x1 -compose Over \
  -coalesce -bordercolor none -frame 4x4+2+2 \
  -bordercolor none -border 2x2 +append script_k_parts.gif
```



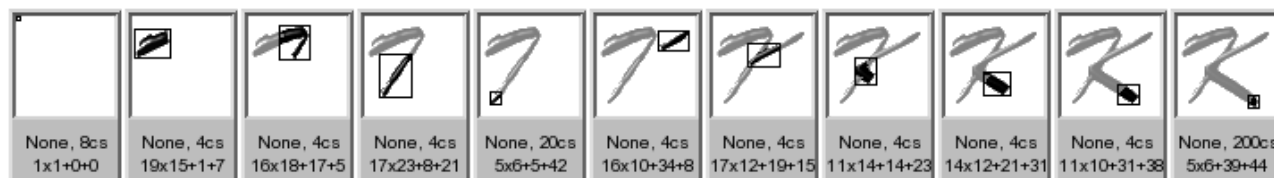
Here you can clearly see how the animation works. Each sub-frame image is positioned so as to add to all the previous overlays. The result is a slowly growing picture. Each frame is also a lot smaller than the 'virtual canvas' on which it is positioned. I use this display technique a lot during the development and debugging of GIF animations, as such I converted it into a shell script "[gif_anim_montage](#)", and expanded it to also list some of the details above each frame in the animation.

```
gif_anim_montage script_k.gif script_k_frames.gif
```



Note the variations in the timings used in various frames, to pause as if the pen is being lifted from the page and repositioned. Animations with variable timing can be some of the most interesting, but also more difficult to handle, as you will see in later IM Example pages. The "[gif_anim_montage](#)" script also the special option '-u' which will also underlay a semi-transparent copy of the coalesced animation. This lets you see how the new sub-frames modifies the displayed animation.

```
gif_anim_montage -u script_k.gif script_k_frames.png
```



Of course this has semi-transparent pixels so a 'PNG' image format was needed, OR you could use one of the many 'background' options that script also provides, allowing you to use GIF or even JPEG formats for the resulting summery image of the animation. Other options, lets you define the number of rows or columns to use, as well as set various non-transparent backgrounds, or use a red box rather than the default black. This script will be used a lot during the next few pages of IM Examples. Suggestions and comments are welcome.

Animation List Information - options used to build an animation

As I noted, using "[+adjoin](#)" and "[-coalesce](#)", as well as "[+repage](#)", are all useful methods of extracting and looking at GIF animations. However they all destroy information about the original animation in the process. You can see this extra information on framing, time delays, frame dispose, etc., using the IM

"[magick identify](#)" command with a "[-verbose](#)" option. However I, and probably most other users, find the output from this command, overwhelming and not really directly usable. This is where another special shell script I wrote comes in. The "[gif2anim](#)" script will separate the individual frames of the animation, but will also figure out exactly what IM "[magick](#)" options you would need in order to re-build the animation from those images. You can think of "[gif2anim](#)" as an animation disassembler, producing a summary of the animation in terms of IM options. For example, lets decode the animation example we have been using to recover the original "[magick](#)" settings used to create it, as well as individual images used...

```
gif2anim canvas_prev.gif
```

```
#
# Animation Sequence File "canvas_prev.gif"
# using a BASENAME of "canvas_prev"
# Extracted on 2007-05-12 17:36:43
#
-loop 0
-page 100x100
-dispose None
      BASENAME_001.gif # 100x100+0+0
-delay 100
-dispose Previous
-page +5+10      BASENAME_002.gif # 32x32+5+10
-page +35+30     BASENAME_003.gif # 32x32+35+30
-page +62+50     BASENAME_004.gif # 32x32+62+50
-page +10+55     BASENAME_005.gif # 32x32+10+55
```



By default the "[gif2anim](#)" script uses the same base file name for the individual images and "[.anim](#)" options file. As such the animation sequence file generated by the above command is named "[canvas_prev.anim](#)", with the individual frame images "[canvas_prev_001.gif](#)" to "[canvas_prev_005.gif](#)". If you examine the results more closely you will see that it actually did manage to re-create the original options I used when I first created this GIF animation (See [Dispose Previous Animation](#)). Also while it is not important to actually generating an animation, the size, and timings of the overlaid frames is also listed as a comment, to make it easier to study. Rather than save the results to a file you can just list the animation sequence options to the screen using a "[-l](#)" flag. That is, just output the animation sequence file, rather than save it, or the individual frame images of the animation.

```
gif2anim -l canvas_prev.gif
```

Given a "[.anim](#)" file and the individual framing images, a complementary script "[anim2gif](#)" can be used to re-build the animation.

```
anim2gif canvas_prev.anim
```



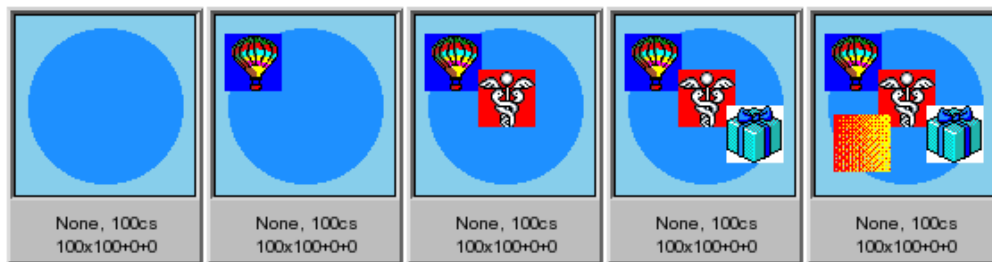
The "[anim2gif](#)" by default will re-create the GIF animation with a "[_anim.gif](#)" suffix. You can see that the resulting "[canvas_prev_anim.gif](#)" animation generated, looks and works exactly like the original animation. This script simply replaces the special string "[BASENAME](#)" used in the "animation sequence file", strips all comments, then just pass the convert options that is left into the "[magick](#)" command. In other words it treats the above file as a type of 'convert' script with comments. The reason a special string was used, is because this then allows you to specify a different base filename than the name of the "[.anim](#)" file itself. That way you can use a completely different set of frame images, such as modified versions of the original,

to recreate a different animation from the old one. This is very useful feature, which will be used in more complex animation processing. (See [Appending Animations Side-By-Side](#) for an example). Like "[gif2anim](#)", the "[anim2gif](#)" script has quite a number of useful options, to help you process and modify animations. Some of these options will be used later. For example see [Appending Animations](#). Also as the ".anim" file is plain text you can use it to take the decoded images of an animation to adjust the GIF's meta-data, such as the timings, positions, repeating sections of an animation, or adding new frames and images to an animation. This was after all why I originally wrote the scripts, long before I got involved with IM examples. For now the "[gif2anim](#)" will be most useful for examining an animation sequence to see just what is happening, and the timings that is being applied between frames.

Dispose Images - the GIF dispose form of the frames

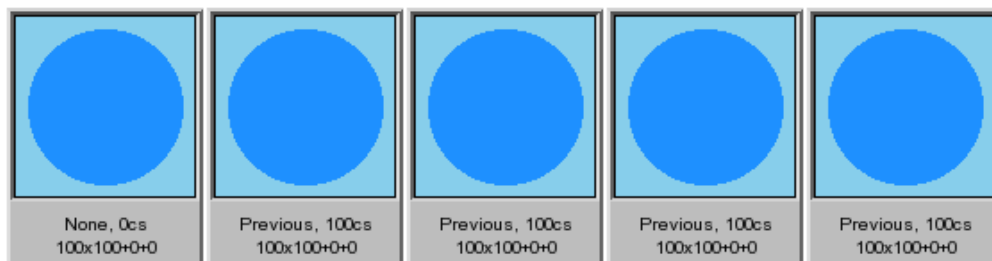
This special "[-layers](#)" method, '**Dispose**' shows what the frame should look like *after* the time delay is finished, and the GIF dispose method has been applied, but *before* the next frames image is overlaid. In other words this shows exactly what the GIF "[-dispose](#)" method setting actually does to the frame, allowing you to figure out exactly what is going wrong with your animation. For example, here is how each of our three [Dispose Method Example Animations](#) look like after the individual frame dispose method was applied. Remember each of these animations consists of a 'canvas image' that was set with a '[None](#)' "[-dispose](#)" setting, then followed by four smaller images overlaid then disposed of by the various GIF dispose methods. '[None](#)' dispose animation...

```
magick canvas_none.gif -layers Dispose canvas_none_dispose.gif
gif_anim_montage canvas_none_dispose.gif canvas_none_dispose_frames.gif
```



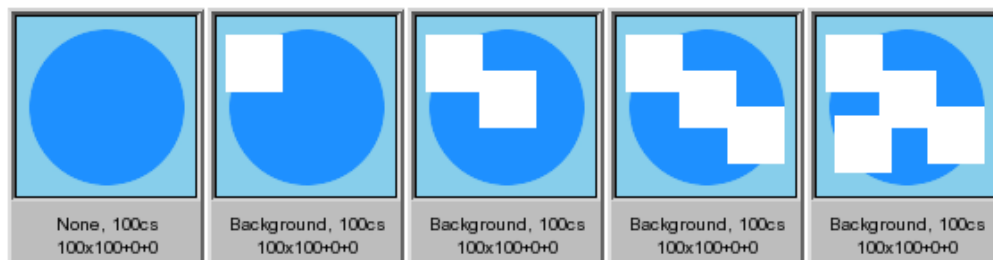
'[Previous](#)' dispose animation...

```
magick canvas_prev.gif -layers Dispose canvas_prev_dispose.gif
gif_anim_montage canvas_prev_dispose.gif canvas_prev_dispose_frames.gif
```



'[Background](#)' dispose animation...

```
magick canvas_bgnd.gif -layers Dispose canvas_bgnd_dispose.gif
gif_anim_montage canvas_bgnd_dispose.gif canvas_bgnd_dispose_frames.gif
```



If you study the above you can see exactly how each of the three GIF dispose methods clear the animation of that frames overlaid image. Note that the first frame of these three animations is always set to a dispose of 'None' so will remain unchanged. It is the effect of the dispose method in the later frames that is important.

 The "-Layers Dispose" operation only generates the 'coalesced' sequence of the disposal frames. It does not reset the disposal setting itself, and as such the result may not animate properly. To make the above animate correctly set all disposal methods to 'previous' or 'background' or you can optimize the animation, before saving.

 The look of the final frame after the GIF dispose method is normally of no consequence to a GIF animation, as the whole canvas is completely cleared before the animation repeats (loops). If it does not 'loop' but stops at the end of the animation sequence, then the final frames disposal is not applied.

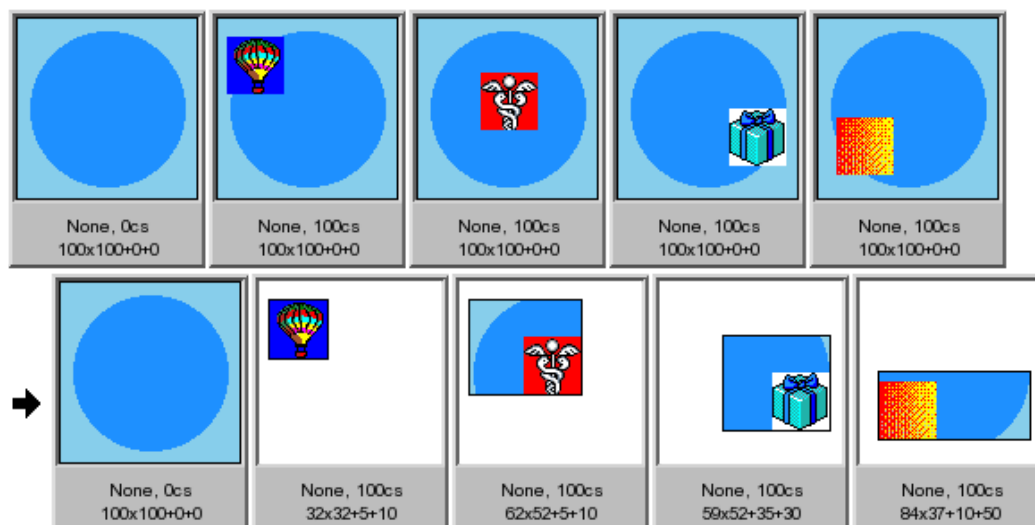
In other words the appearance of the last frame (after disposal) as shown above, or even the actual dispose setting of the last frame, does have any effect on a GIF animation. IM generally sets this to same as the previous frame when it tries to work out an appropriate disposal method, during a [Frame Optimize](#) an animation.

Deconstruct - report areas of frame differences

The traditional way in ImageMagick to optimize an animation, making the result smaller and faster to download and animate, is to "[-deconstruct](#)" its "[-coalesce](#)" form. *This is no longer recommended.* Instead you use should use the [General GIF Optimizer](#). This operator will take a coalesced sequence of images (the animation frames as they actually appear when displayed), and compare the second and later images with that of the previous image. It then replaces that image with the smallest rectangular area of the pixels that had changed. Any pixel change, will count, regardless of if it is a color change (overlay) or cleared (erased). This is quite simple, and for a typical [Overlay Animations](#) will generate an optimal [Frame Optimization](#) for that animation. An [Overlay Animations](#) animation however only use a 'None' dispose method only. For example lets take the [coalesce previous animation](#) we generated above, which happens to form a [Overlay Animation](#), and run it though the "[-deconstruct](#)" operator.

```
magick canvas_prev.gif -coalesce coalesce.gif
magick coalesce.gif -deconstruct deconstruct.gif
gif_anim_montage coalesce.gif coalesce_frames.gif
gif_anim_montage deconstruct.gif deconstruct_frames.gif
```





A 'previous dispose animation' if you remember clears each frame to the last non-previous disposed frame, in this case the initial background canvas. As you can see "[-deconstruct](#)" returned the area that changed from one coalesced frame to the next. Resulting in an optimized [Overlay Animation](#), that requires no special dispose setting. This is nowhere near as optimal as the original hand generated animation I started with, but is useful in itself.

Unfortunately "[-deconstruct](#)" has absolutely no understanding of the GIF animation "[-dispose](#)" settings. Consequently if you try this on an animation that clears pixels from one frame to the next, such as the '[background disposed](#)' animation we created above (and shown left), it will fail badly.

Here we take the just shown animation, and run it through a "[-coalesce](#)" and "[-deconstruct](#)" cycle.

```
magick canvas_bgnd.gif -coalesce -deconstruct deconstruct_erase.gif
```



As you can see "[-deconstruct](#)", slowly destroys the animation. Basically "[-deconstruct](#)" is designed to simply find the differences between image layers. It was never designed to correctly optimize animations, and will fail for animations that need to use various disposal techniques to clear (erase or make transparent) previously overlaid pixels.

Frame Comparisons - more detailed comparing of frames

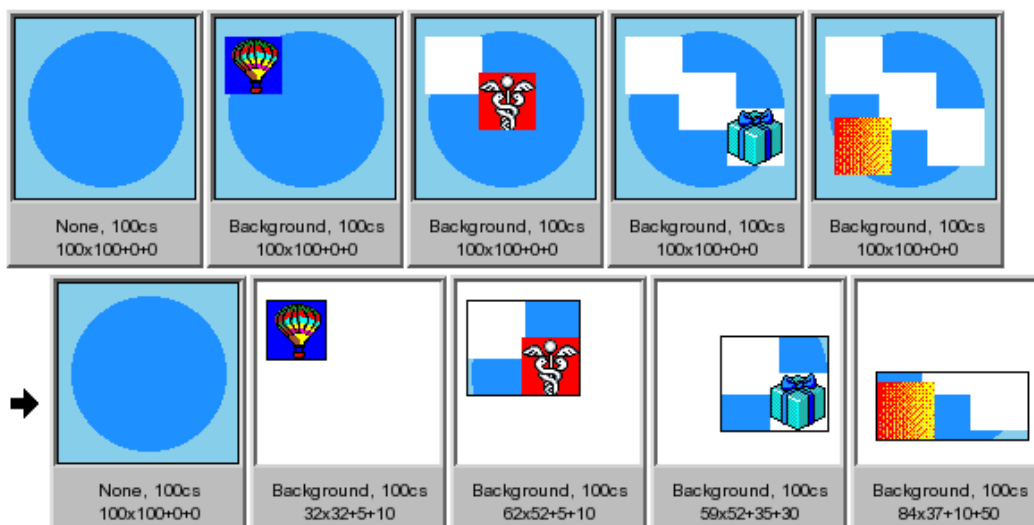
With IM v6.2.6-2, a number of extra GIF frame comparison methods were added. These were needed internally for proper optimization of animations, but was deemed useful enough to make them available to the command line and other API interfaces.

Compare_Any

The "[-layers](#)" method '[CompareAny](#)' is actually exactly the same as "[-deconstruct](#)". In fact the "[-deconstruct](#)" operator is only a functional alias for the '[CompareAny](#)' method. Again let's look at the actual image results of a 'deconstruct' or '[CompareAny](#)' of the '[background disposed](#)' animation.

```
magick canvas_bgnd.gif -coalesce canvas_bgnd_coal.gif
gif_anim_montage canvas_bgnd_coal.gif canvas_bgnd_coal_frames.gif

magick canvas_bgnd_coal.gif -layers CompareAny magick compare_any.gif
gif_anim_montage compare_any.gif compare_any_frames.gif
```

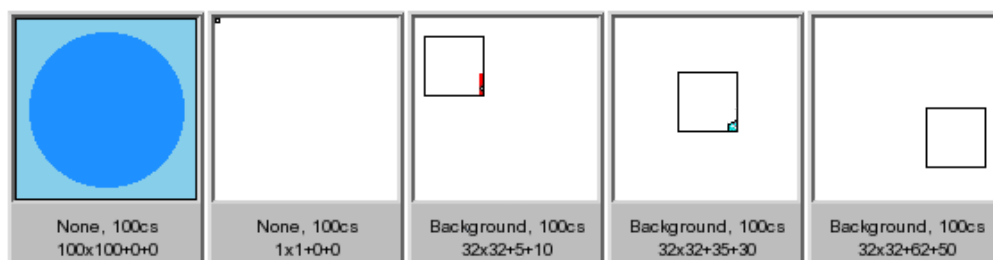


As you can see the second and later images, is the minimal rectangular area that contains all the pixels that have changed, whether it is an overlay of a new pixel color, or a clearing of an old pixel to transparency.

Compare_Clear

The "[-layers](#)" method '[CompareClear](#)' will show the smallest rectangular area that contains all the pixels that needed to be cleared from one frame to the next.

```
magick canvas_bgnd_coal.gif -quiet -layers CompareClear compare_clear.gif
gif_anim_montage compare_clear.gif compare_clear_frames.gif
```

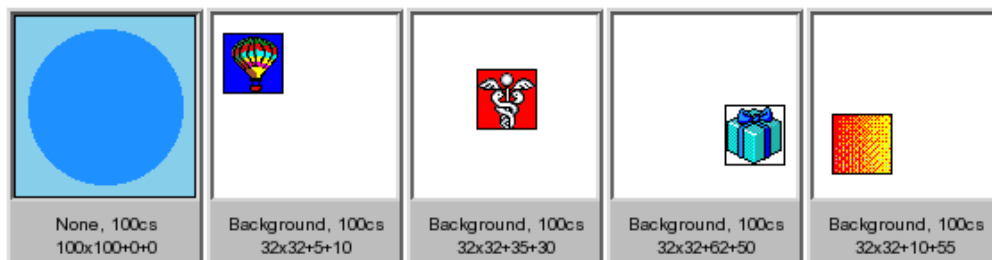


Notice that as no pixels were cleared between the first and second frame, a special [Missed Image](#) was generated. The "[-quiet](#)" setting was used to tell IM not to give any warning about this image. If all the later frames all become 'missed' images, then the GIF animation never clears pixels, and the animation can be classed as a [Overlay Animation](#).

Compare_Overlay

The last "[-layers](#)" comparison method, '[CompareOverlay](#)', returns the area of pixels that were overlaid (added or changed in color, but not cleared) since the previous frame.


```
magick canvas_bgnd_coal.gif -layers CompareOverlay magick compare_overlay.gif
gif_anim_montage compare_overlay.gif compare_overlay_frames.gif
```



This is similar to the special IM specific '[ChangeMask](#)' Alpha Composition Method. However, that returns just the pixels that change the image, rather than the rectangular area that was changed. See also [Transparency Optimization](#).

 None of the "[-Layers](#)" comparison methods, nor the "[-deconstruct](#)" operator, look at or modify the image GIF dispose method that is used. The results is just a list of images, and not expected to be used as animations themselves.

 While the operators are designed to work with a coalesced image sequence, they will accept a non-coalesced sequence of image layers, without producing an error.

In this case each frame is overlaid onto the previous overlaid frames using a '[Copy](#)' alpha composition method, before the frames are compared. This alpha composition method ensures that any transparency in a layer will also be added to the destination image. Without this the above would not find pixels that get cleared to transparency in a coalesced image sequence.

Note that this is different to the more normal '[Over](#)' composition method that the "[-coalesce](#)" operator would use for handling the 'dispose/overlay' cycle needed to display a GIF animation.

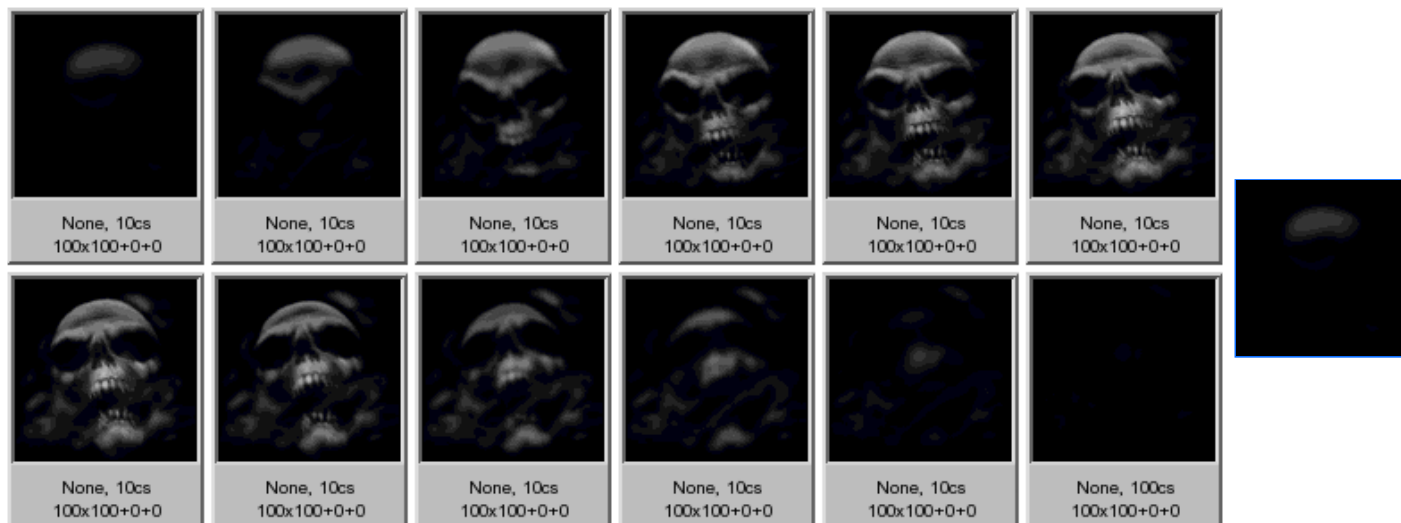
Types of Animations

Most GIF animations you find fall into some basic types of animation. Knowing about these types allows you understand how that animation is being displayed from one frame to another, and can allow you to take shortcuts in how you handle and modify the animation.

Coalesced Animations

A '**Coalesced Animation**' is basically an image sequence that shows what an animation should look like when displayed to an user after each 'dispose/overlay' cycle. The images are basically as you would see them if you were looking at an actual 'film strip' of the animation. It is the simplified and completely un-optimized form of any animation. This naming convention is from the name of the IM "[-coalesce](#)" operator that is used to magick GIF disposal animations, into an un-optimized 'Coalesced Animation'. Most video formats (MPEG, AVI etc) are actually also 'Coalesced Animations' by their very nature. However these also tend not to have any transparent pixels, and generally have a constant time delay between the frames of the animation. A coalesced GIF animation however can have transparent pixels, and widely varying time delays from immediate '0' delays, to very fast, or very very very slow. Any GIF disposal setting in a coalesced animation does not have any meaning, however the "[-coalesce](#)" operator will set the disposal appropriately so the resulting image sequence can still work as a valid GIF animation. Video formats which always

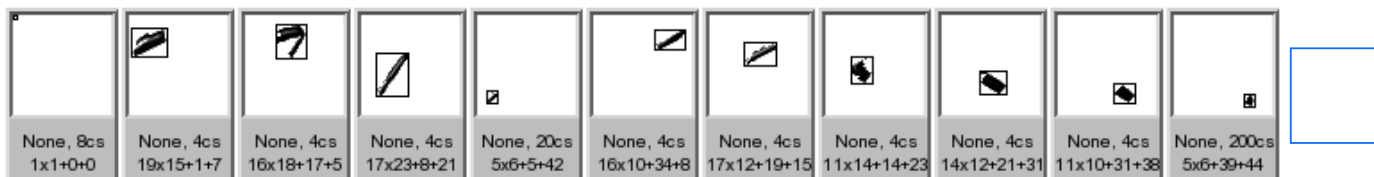
replaces every pixel from one frame to the next can generally just use a '[None](#)' or '[Undefined](#)' GIF disposal setting. Here is an example of an animation that is also by its nature a Coalesced Animation, plus a "[gif_anim_montage](#)" display of the animations individual frames.



Most animations that do not contain, or use, transparency, and which animate the entire canvas, are usually saved and distributed as Coalesced Animations.

Overlay Animations

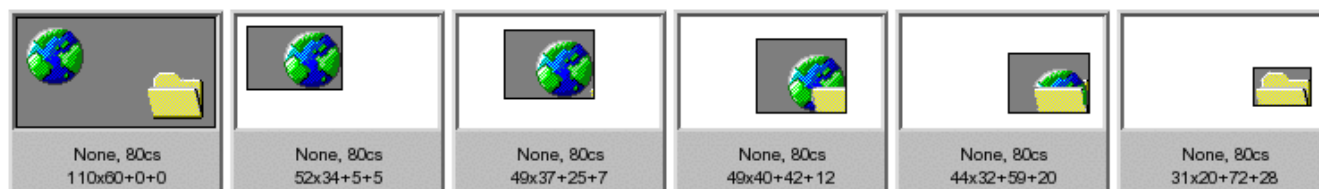
An '**Overlay Animation**' is one in which each frame of an animation only overlays new pixels to the animation currently displayed. In other words at no point in the animation does it need to clear a pixel to transparency. The individual frames can contain transparency, either as a background, or as part of its optimization, but it never clears a pixel back to transparency. Of course if no transparency is used at all, then the animation is guaranteed to be able to be turned into a simple Overlay Animation. This is probably the simplest type of [Frame Optimized](#) animation, and one that requires no special handling by clients. Each frame that is shown to the user can be viewed as simply as 'flattened' image of all the previous frames. Any animation that only uses a '[None](#)' GIF disposal method is an 'Overlay Animation'. The last example for instance is not only a '[Fully Coalesced Animation](#)' but also a 'Overlay Animation', though not all such 'Fully Coalesced Animations' are 'Overlay Animations'. You can test if an animation can become an overlay animation by using the '[CompareClear](#)' layers method on a [Coalesced](#) animation, and checking if all the second and later images are 'missed images'. That is, no pixel needed to be cleared or erased from one frame to the next. In fact if an animation can become an overlay animation, without modification, then the IM "[coalesce](#)" operator will only use '[None](#)' disposal methods for all the frames. If this is not the case then "[coalesce](#)" will have used '[Background](#)' disposal for at least some of the frames. This then gives you with another test for 'overlay only' capability. Overlay animations can use transparency, but they do not have any moving parts on a transparent background. For example, the animation of a hand drawn letter 'K', is an overlay animation, as each part only adds or changes existing parts on a transparent background. It never adds new transparency to the resulting image (except as part of the first frame).



That is, not to say an moving object can't be handled by an overlay animation, it just means you need a non-transparent background so you can also 'erase' the old positions



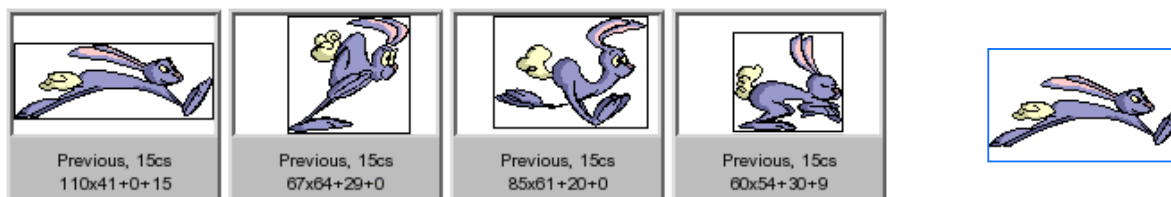
of the moving parts without needing transparency. For example look at the frames of this "download the world into a folder" animation...



Of course by needing to 'erase' old parts by overlay the original background means that overlaid sub-images are generally bigger, and hence a generally larger GIF animation file size. Unfortunately using the IM's [Optimize Frame](#) operator, can, and most probably will, turn a 'Coalesced Overlay Animation' into into something that is not a 'Overlay Animation', in its attempt to find smaller GIF file size. However by using [Deconstruct](#) on the animation instead of using [Optimize Frame](#), you can ensure the animation remains a simple 'Overlay Animation', but only if the animation is really a 'Overlay Animation'. If the animation isn't a 'Overlay Animation', then [Deconstruct](#) operation can go badly wrong (See [Deconstruct](#) above). With some human skill you can still optimize an overlay animation better, for example using [Split Frame Updates](#), and applying some form of [Compression Optimization](#) without destroying the 'Overlay Only' requirement of the animation. Typically 'Overlay Animations' display no transparency at all (they can use it as part of optimization, but they don't display it). And if no transparency is displayed, then the animation is guaranteed to be a 'Overlay Animation'. Why is 'Overlay Animations' so important? Because there is software out there that are limited to this type of animation. It is much simpler to handle as only overlay is performed without need to handle transparency, or save the previous frame to handle GIF disposal methods. Such software is rare but does exist.

Cleared Frame Animations

When an animation only uses just '[Previous](#)' or '[Background](#)' GIF disposal, you get a very special type of animation. Note that if only '[Background](#)' disposals is used, all the frames of an animation are displayed then cleared before the next frame is displayed. Also when only '[Previous](#)' are used, the animation is always returned to the initial cleared canvas before the next frame is displayed, resulting in the same effect. Same happens if you only use a mix of those two disposal settings. That is, animations that only uses these disposal methods will have frames that are complete copies of what is to be displayed. That is, the frame contains everything that will be displayed to the user at that point in time. That is, not to say that the animation is a '[Fully Coalesced Animation](#)'. As the sub-frame may be a lot smaller than the virtual canvas area of the animation, but everything outside that frame will be regarded as transparent (or background), containing nothing of importance. For example take a look at this running bunny animation...



Note that each and every sub-frame is the complete image that is displayed. No more, no less. Also notice that none of the frames actually needs to use the whole virtual canvas of the animation. And finally note how all the frame disposals are set to '[Previous](#)', which for reasons you will see below is the more logical disposal setting to use. All the disposal settings could however have been set to '[Background](#)' disposal or any mix of the two without changing the final result. For want of a better name I call such an animation a '**Cleared Frame Animation**', but I have also seen it called a 'Previous or Background Disposal Animation'. The only time that an animation is not of this type is if at least one non-blank frame in the animation used

a '[None](#)' or '[Undefined](#)' (same thing) disposal method. This animation is very special, as it can handle any amount of transparency clearing, anywhere within the animation sequence, unlike a [Overlay Animation](#). But it can also be overlaid onto ANY static background image really quickly. To do this we need to tighten the definition of a 'Cleared Frame Animation' slightly. Specifically we need to make sure all the disposals are set to '[Previous](#)' (which is already the case in our example). If that is done, then you can just pre-pend an image (with a zero delay) to underlay a background.

For example lets place our bunny on some grass....

```
magick bunny_grass.gif bunny_anim.gif -loop 0 bunny_on_grass.gif
```



As you can see this is so simple that many applications use these types of GIF animations to add symbols or other indicators (file locks, smileys, stars, etc) to larger objects. Animating such a GIF animation is also easy, as the application can just clear the area to some simple constant background image, and overlay the next frame in the sequence. No need to calculate disposals, or keep track of a 'previous' display, other than the static unchanging background display. This is also the reason why a '[Previous](#)' disposal is the preferred disposal for a [Cleared Frame Animation](#). Unlike a [Overlay Animation](#) which are only a special sub-set of GIF animations, ALL animations can be saved as a [Cleared Frame Animation](#). Just [coalesce](#) the animation, and optionally [trim](#) any surrounding transparent edges to frame optimize it, and reset the disposals.

```
magick any_animation.gif -coalesce -trim \
    -set dispose previous cleared_frame_animation.gif
```

You can even re-position the animation on that background...

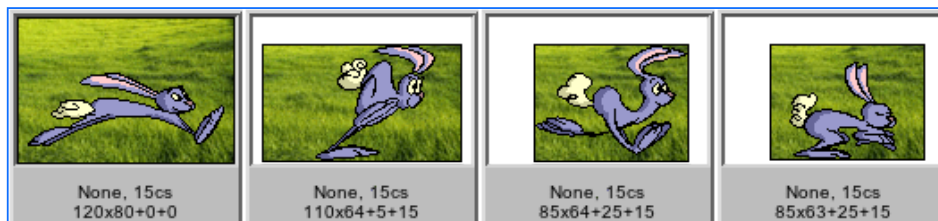
```
magick bunny_grass.gif \( bunny_anim.gif -repage 0x0+5+15\! \) \
    -loop 0 bunny_on_grass2.gif
```



As such a [Cleared Frame Animation](#) typically consists of a small, constantly changing or moving object on a transparent background. These are directly usable on web pages, or as animated symbols, or can be merged with other animations to produce much more complex animations. In summery this type of animation is a good style to use in a library of animated parts, for use in creating larger more complex animations.

There is however a problem with adding background like this for GIF animations. If you look at the previous examples, you would probably have notice a significant and disturbing pause in the fast moving animation. That is, the bunny vanished for a moment, when the animation looped, and the background image is re-loaded. (See notes in [Zero Delay Frames](#)) Though this is not a problem for applications that use this technique, for adding animated symbols to displayed objects, as they just don't display that 'intermediate' frame, anyway. As such if are adding a non-transparent background to the GIF animation, then it is generally a good idea to magick the simple [Cleared Frame Animation](#) into a [Overlay Animation](#). That is add that background to every frame of the animation, rather than give an initial canvas. You can do that by either, [Coalescing](#) the above animation, then deleting the [Zero Delay](#) Background frame, OR [Layer Composite](#) the original animation onto a [Static Background](#). For example...

```
magick bunny_grass.gif \( bunny_anim.gif -repage 0x0+5+15\! \) \
    -coalesce -delete 0 -deconstruct -loop 0 bunny_bgnd.gif
gif_anim_montage bunny_bgnd.gif bunny_bgnd_frames.gif
```



Now that all the frames are displayed equally well, no pause can be seen as the background is reset. The animation however is now an [Overlay Animation](#) with background being 're-drawn' due to the movement of the animation object, as such it is probably a bit larger in file size. See [Transparency Optimization](#) for a continuation of the optimization of the above result.

 The IM forum member [el supremo](#), Pete, has contributed a MagickWand equivalent script, [Cleared Frame onto Background example](#).

This example is also discussed in detail in the IM Forum [Creating a Cleared Frame GIF Animation in the MagickWand](#).

Mixed Disposal Animations - multi-background animations

There is nothing preventing you from mixing the various disposal methods in a single GIF animation. In fact adding a background to a [Cleared Frame Animation](#), does exactly that. Using mixing disposal methods not so simple for us humans, but doing so can allow you to generate some very complex animated displays. In general they are created as part of automatic [Frame Optimization](#) for a particular animation. Just remember the result will not be directly usable as the previous animation types for specific purposes. In fact do not be surprised if some GIF handling programs, just do not handle a 'Mixed Disposal Animation' properly. That includes some of the GIF optimizers available. A typical example of a 'Mixed Disposal Animation' is a small moving object that causes some semi-permanent but temporarily static change in the animations background. A ball hitting a lever would be one example.

Simple Example wanted

Similarly animations involving two very small moving objects on a larger transparent display can only be optimized well by mixing up the disposal techniques, so that each object is moved using separate animation frames.

FUTURE,

A more complex animation for study.

- * Start with a semi-transparent canvas
- * run a little 'previous' disposal
- * leave one frame as a new 'canvas'
- * more previous animations
- * erase that 'addition' using a 'background' disposal set as new canvas
- * continue back to start point.

This animation should thoroughly test out not only IM disposal methods but also various browser disposal methods.

If you like to contribute an IM example of such an animation you can get your name, and home page link here!

The End of the Loop - when an animation stops

It is often regarded as a good idea not to make animations loop forever, as client machines have to continually work while the animation is still animating. As such it is a good idea to think about how many times your animation actually loops. Basically..

Add a loop limit to your animations (if practical)

This is especially the case for large animations used as a logo at the top of a web page. Depending on the overall time for which an animation runs, 10 to 30 loops is usually enough for large logos. And you should add a loop limit for all such animations, to be kind to your users. A "[-loop](#)" save setting of 0, means to loop forever. This is typically what is used, and that is okay, for small animations. Some browsers however enforce a stop when an animation reaches some internal loop limit (often 256 loops). For IM Examples I have generally used a '0' "[-loop](#)" save setting, meaning 'loop forever', because the animations can appear anywhere on a page. That means it may be some time between when the user loaded a page to when they finally look at and read about a particular GIF animation. If I did use a smaller number of 'loops', the animation would no longer be doing what it should be demonstrating, losing its effectiveness. For large animations on top level flash pages, a "[-loop](#)" of '1', the normal GIF animation default, can be used. This means run though the animation once only and then stop. Which brings us to an all important question..

Stop where? Some browsers, like the old "[Netscape](#)" browser, re-displayed the first frame of the animation. Most modern browsers however will just stop on the last frame, ignoring that frames useless [dispose](#) setting. What a specific application does however is up to the application as there is no true standard. Actually even the use of the 'loop' meta-data, is itself non-standard, just something that the old "[Netscape](#)" browser implemented, and all later browsers copied. Because of this, if there is some specific frame that you want your animation to stop on, say the frame with your company name or logo on it, then it is a good idea to make that frame both the first and last frame of an animation. One of those frames however should be given a 'zero delay', so as not to effect the overall loop time length of the animation. So just to summarize...

If loop limited, add the 'stop' frame as BOTH first and last frames.

Zero Delay Intermediate Frames

We have already seen use of a frame that has a 'zero delay', with regard to a [Cleared Frame Animations](#). I also used them to explain [Previous and Background Disposals](#). These special frames are actually a lot more common then people would probably think. They represent not only methods to prepend a background 'canvas' to an animation (such as I have used them for above). But they are also a must for some of the more complex [Frame Optimization](#) techniques, such as [Frame Doubling](#) and [Splitting Frame Updates](#). Another (very old pre-PNG format) usage was to allow you to create static GIF images that contained more than the 256 color limit! That is, each frame provides 256 colors, and the next frame the next set of 256 colors, all with zero delay and no looping at the end. Thanks to [TLUL](#) in the discussion [Creating unquantized GIFs](#) for pointing this out. These 'zero delay intermediate frames' are not meant to be displayed to an user. They are just used to create special effects in GIF images that are otherwise not possible or better optimized than you would get without them. In summary...

Zero Delay Frames are Intermediate Frames, They are not meant to be visible to users.

ImageMagick not only will create such frame in animations as part of its automatic '[OptimizePlus](#)', but also provides a way to remove them using the '[RemoveZero](#)' layers method. Watch out for them, as they will often complicate your handling of Animations. Okay so they are important, what of it? Because many applications don't like them, or incorrectly handle them. They consider 'Zero Delay Frames' as a bad thing, even when you purposely add them to animations, for some reason or another. Here is summary of applications that I know about or have been told, 'do the wrong thing'...

Gimp Will not save a 'Zero Delay Frame', they always add a minimal time delay to any frame that has a zero time delay. :-(

Firefox Will give a slight non-zero pause on such frames. This presumably is so that animations that have no time delays at all, do use up all the computers CPU cycles. But "**firefox**" still doesn't relax that restriction if an animation has an overall non-zero display time.

Internet Explorer Has a minimum time delay of 6 centi-seconds, and ignores any delay smaller than this. Internet Explorer version 8 also fails (immediately restarts the loop) if any image frame extends beyond the animation bounds set by the first frame. This I would class as a major bug.

On the other hand ImageMagick '[RemoveZero](#)' layers method does do the right thing and will NOT remove any frame if ALL the images have a 'zero time delay'. In fact this layers method will give a warning if it sees an animation with no time delays at all. This brings us to another rule-of-thumb...

Never save a GIF (looping) animation that has no 'delays' at all

Doing so is very bad practice, and the reason most of the above 'buggy' applications, do what they do, rather than what they should. Complain to the owners if you see them. Also complain to application developers, so that they handle Zero delay frames correctly. Even if it means not displaying that frame at all, just using them as preparation for the next frame to display. They are after all on screen for *ZERO* time!

Created: 24 July 2004 (sub-division of "animation")

Updated: 27 February 2013

Author: [Anthony Thyssen](#), <Anthony.Thyssen@gmail.com>

Examples Generated with:

URL: https://imagemagick.org/Usage/anim_basics/